Eurographics/ IEEE-VGTC Symposium on Visualization (2009), pp. 1–000
H.-C. Hege, I. Hotz, and T. Munzner (Guest Editors)

Reactive Workflows for Visual Analytics

Ioana Manolescu[1] and Wael Khemiri[2] and Véronique Benzaken[2] and Jean-Daniel Fekete[1]

[1]INRIA, France, [2]Univ. Paris-Sud, France

An important class of visual analytics applications has to deal with *dynamic data*, which is continuously updated (e.g. by receiving new additions) *while* the analysis process is running. For instance, a visual analysis of Wikipedia articles, based on a live feed of Wikipedia data. The analysis has to gracefully adjust to the updates; stopping the process and triggering a total re-computation is not acceptable.

We have designed ReaViz, a *reactive workflow platform*, conceived and deployed in close connection with a database of application and workflow-related data. ReaViz enables the declarative specification of data-driven workflows, enforcing a clean separation between the process data, the process structure, and the user interaction which may or may not involve visualization. A distinguishing feature of ReaViz is the possibility provided to the process designer to specify *reactive behavior* in the event of data changes to one or several of the data collections involved in a process. Thus, ReaViz aims at enabling the easy, elegant specification of visual analytics applications where data dynamics is an important aspect.

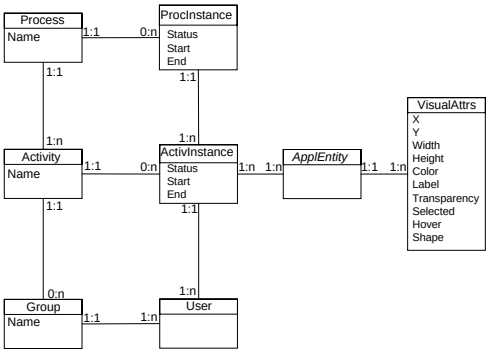

Figure 1: *ReaViz data model.*

1. Related works

Research in data visualization has produced several interactive platforms for data visualization [**?**]. Such platforms focus on the interaction between the human expert and a data set consisting of a completely known set of values, but they do not facilitate the inclusion of data analysis programs on the data.

In parallel, significant research and development efforts have been spent establishing models [**?**, **?**, **?**] and platforms for workflow specification and deployment. In recent years, moreover, *scientific workflow* platforms have received significant attention [**?**]. Such platforms are based on sligthly different premises than regular (business-oriented) workflows. Most notably, scientific workflows incorporate data analysis programs (or scientific computations more generally) as a native ingredient, and are meant to be specified by scientists, their end users. This contrats with business workflows, typically specified by some business analysts and enacted by actors which do not need to have a global view of the process.

Scientific workflows have enabled the development of numerous applications. Thus, one could attempt to implement visual analytics applications as instances of scientific processes. However, the platforms we have surveyed have several shortcomings. First, the relationship between the data and the process specification is not well formalized. Second they are not easily extended to handle dynamic data sources, while preserving clear semantics.

2. A model and tool for reactive processes

2.1. Data model

The data model of a ReaViz application is a set of relations, which can be classified as belonging to three categories. *Application-dependent* relations model the data used by the specific visualization/analysis application. Separately, a set of *workflow-related* relations are used to capture the definition and instances of workflows. Finally, *visualization-related* relations capture the information items required by

the data visualization modules. An outline of the data model is provided in Figure **??**. On the left, the entities *Process*, *Activity* and *Group* describe the process schemas (or process descriptions); the latter entity corresponds to groups of users. The next vertical colum comprises entities used for process instance bookkeeping, namely: process instance, activity instance, and specific users which belong to groups. An activity instance has a start date and an end date, as well as a *status* flag which can take the values: *not_started*, *running* and *terminated*. The status of a process instance can take similar values.

The *ApplEntity* entity refers in a generic manner to all entities which may actually be used by a given application. The relationship between *ApplEntity* and *ActivityInstance* captures is to be instantiated according to the specifics of each application; it represents the way activity instances are created or otherwise manipulated during process instance execution.

Finally, the entity *VisualAttributes* encapsulates a set of attributes frequently used in data visualisation applications, such as: (x,y) coordinates, width, height, color, label (a string), transparency, whether the data instance is currently *selected* by a given visualisation component etc.

2.2. Process model

We consider a simple yet expressive model for describing reactive processes. A *reactive process* is defined as a 5-tuple consisting of:

- a set of relations or queries R resulting from a data model;
- a set of variables, where each variable v is a name, value pair;
- a set of procedures, each procedure p being a computation unit implemented by some external, black-box software (developed outside the workflow engine by means in a language such as C++, Java or MatLab) that takes relations as parameters;
- a process P; and
- a set of *compensating actions* CA.

For the sake of conciseness, we leave out the formal specifications of these entities except for the reactive process:

$$RP ::= R^*, v^*, p^*, P, CA^* \quad (1)$$

A *compensating action* CA specifies what should be done in the event that a set of tuples, denoted ΔR, are added to an application-dependent relation R *during the execution of a process instance*. Observe that we consider additions to R, regardless of the way the addition takes place: in particular, it may be due to a table assignment on R (thus, take place within the enactment of our processes), but it may also be due to some external event which modified R.

Let $t_{\Delta R}$ be the moment when ΔR was received. Several options are possible.

1. Ignore ΔR for the execution of all *processes* which had started executing before $t_{\Delta R}$. The data will be added to R, but will only be visible for process instances having started after $t_{\Delta R}$. This recalls a process-granularity transactional model, where each process operates on exactly the data which was available when the process started. This is the default behavior.
2. Ignore ΔR for the execution of all *activities* which had started executing (whether they are finished or not) before $t_{\Delta R}$. However, for a process already started, instances of a specific activity which start after $t_{\Delta R}$ may also use this data.
3. As a macro over the previous option and the process structure, one could wish for ΔR to be propagated to instances of all activities that are yet to be started in a running process.
4. Execute compensation work for all the terminated instances of a given activity. We can moreover specialize the behavior on whether we consider only activity instances whose process instances have terminated, only activity instances whose process instances are still running, or both.
5. Execute compensation work for all the running instances of a given activity. Obviously, in this case, the process instances to which the activity instances belong are all running.

3. Conclusion

We believe this formal approach will lead to cleaner and more scalable visual analytics applications, built upon well understood and reliably implemented components such as databases and workflow systems.

We need to continue integrating existing analysis and visualization components into this framework to better understand the implications of supporting reactive workflows for existing implementations.